DERWENT-ACC-NO: 2001-366857

DERWENT-WEEK: 200140

TITLE: Task parallelization method in parallelizing compiler for multiprocessor system, involves executing information transfer task with respect to next execution task, by idle processor to which task is not allocated

INVENTOR: AOKI, Y; SATO, M

PATENT-ASSIGNEE: HITACHI LTD[HITA] , AOKI Y[AOKII] , SATO M[SATOI]

PRIORITY-DATA: 1999JP-0347090 (December 7, 1999)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 20010003187 A1	June 7, 2001	N/A	025	G06F 009/00
JP 2001167060 A	June 22, 2001	N/A	017	G06F 015/16

APPLICATION-DATA:

PUB-NO	APPL-DESCRIPTOR	APPL-NO	APPL-DATE
US20010003187A1	N/A	2000US-0729975	December 6, 2000

JP2001167060A	N/A	1999JP-0347090	December 7, 1999

INT-CL (IPC): G06F009/00, G06F009/45 , G06F015/16

ABSTRACTED-PUB-NO: US20010003187A

BASIC-ABSTRACT:

NOVELTY - Data regarding relevant task satisfying preset condition and instruction code in the task are detected. Information transfer task that is produced to instruct the storage of detected data in storage apparatus closer to processor to which task is allocated, is added with task scheduling process. Transfer task is executed with respect to next execution task by idle processor to which task is not allocated.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the following:

(a) Task parallelization apparatus;

(b) Computer readable storage medium which stores program used to execute task parallelization

USE - For converting source program into specified program and object code, in parallelizing compiler of multiprocessor system.

ADVANTAGE - Even when total number of executable task is smaller than the total number of usable processors at certain time instant while the program is executed, either the program or the object code is output in short time, by efficiently utilizing idle processor to which task is not allocated, and hence the performance of the multiprocessor system is improved.

DESCRIPTION OF DRAWING(S) - The figure shows the parallelizing compiler.

CHOSEN-DRAWING: Dwg.3/15

TITLE-TERMS: TASK METHOD COMPILE MULTIPROCESSOR SYSTEM EXECUTE INFORMATION TRANSFER TASK RESPECT EXECUTE TASK IDLE PROCESSOR TASK ALLOCATE

DERWENT-CLASS: T01

EPI-CODES: T01-F02A; T01-F02C; T01-F03B; T01-F05A; T01-S03;

SECONDARY-ACC-NO:

Non-CPI Secondary Accession Numbers: N2001-267670

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2001－167060
（P2001－167060A）
(43)公開日　平成13年６月22日(2001.6.22)

(51)Int.Cl.'	識別記号	ＦＩ		ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　15/16	６３０	Ｇ０６Ｆ　15/16	６３０Ｃ	５Ｂ０４５
9/45		9/44	３２２Ｆ	５Ｂ０８１

審査請求　未請求　請求項の数１　ＯＬ　（全 17 頁）

(21)出願番号　特願平11－347090

(22)出願日　平成11年12月７日(1999.12.7)

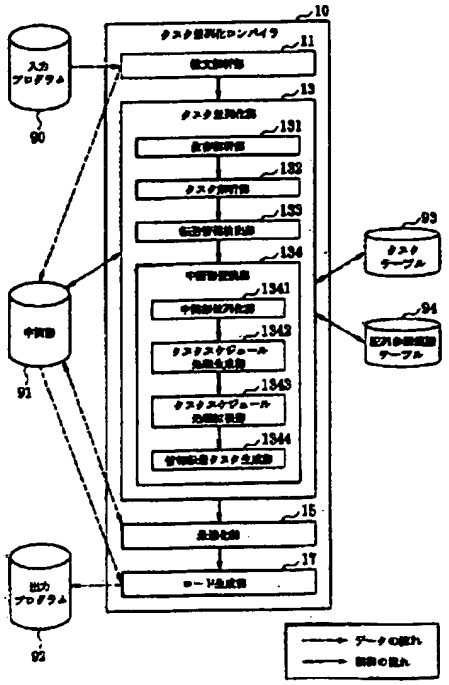
（出願人による申告）産業再生法第30条の規定による特定研究成果に係る特許を受けようとする出願

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地
(72)発明者　青木　雄一郎
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内
(72)発明者　佐藤　真琴
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内
(74)復代理人　100102587
弁理士　渡邉　昌幸　（外１名）

Ｆターム(参考)　5B045 GG11
5B081 CC32 CC41

(54)【発明の名称】　タスク並列化方法

(57)【要約】
【課題】　従来は、実行可能なタスク数が利用可能なプロセッサ数より少ない場合に発生するアイドルプロセッサを有効利用できない。
【解決手段】　タスク内で参照される可能性のあるデータまたはタスクに含まれる命令コードをコンパイル時に検出し、該データまたは命令コードを、タスクが割り当てられるプロセッサに近い記憶装置へ転送する命令からなる情報転送タスクを生成し、タスクを実行していないアイドルプロセッサに次に割り当てる次実行タスクとして各プロセッサで実行中のタスクで最も早く終了するタスクを求め、この次実行タスクに対する情報転送タスクがアイドルプロセッサで実行されるよう割り当てる命令からなる情報転送タスクスケジュール処理を、並列コンパイラが生成するタスクスケジュール処理に追加する。



1

【特許請求の範囲】

【請求項1】　ソースプログラムを、並列計算機で実行可能な複数のタスクと該タスクをプロセッサへ割り当てるタスクスケジュール処理とからなるプログラムもしくはオブジェクトコードに変換する並列化コンパイラにおけるタスクの並列化方法であって、予め定められた条件を満たすタスクA内で参照される可能性のあるデータならびに上記タスクAに含まれる命令コードをコンパイル時に検出するステップと、上記データならびに上記命令コードを、上記タスクAが割り当てられるプロセッサに近い記憶装置へ転送する命令からなる情報転送タスクを生成するステップと、タスクを実行していないアイドルプロセッサに次に割り当てる次実行タスクを求めて該次実行タスクに対する上記情報転送タスクが上記アイドルプロセッサで実行されるよう割り当てる命令からなる情報転送タスクスケジュール処理を上記タスクスケジュール処理に追加するステップとを有することを特徴とするタスク並列化方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、ソースプログラムをタスクに分割して、並列計算機で実行可能なプログラムまたはオブジェクトコードに翻訳・変換する並列化コンパイラのタスク並列化技術に関わり、特に、高速実行可能なプログラムまたはオブジェクトコードを出力するのに好適なタスク並列化方法に関するものである。

2

*【0002】

【従来の技術】従来、並列化コンパイラにおけるタスクの並列化は、例えば、合田憲人、岩崎清、岡本雅巳、笠原博徳、成田誠之助　著「共有メモリ型マルチプロセッサシステム上でのＦｏｒｔｒａｎ粗粒度タスク並列処理の性能評価」情報処理学会論文誌、１９６６年３月号、Vol.３７、No.３、４１８－４２９ページ(以降、「文献１」と記載)で述べられているように、次の３つのステップから構成される。

【0003】（１）プログラムをタスクと呼ばれる小部分に分割する。

（２）タスク間の制御の流れ、変数の参照順序関係から、各タスクの「実行可能条件」を導出する。

（３）タスクおよびタスク先頭に挿入した実行可能条件を含む「タスクスケジュール処理」から構成されるプログラムまたはオブジェクトコードを生成する。

【0004】ここで、「実行可能条件」とは、タスク間の実行順序関係を表した条件であり、この条件を満たしたタスクは実行開始してよいことを意味する。また、「タスクスケジュール処理」とは、タスクを実行していないアイドルプロセッサの有無とタスクの実行可能条件の成立を監視し、実行可能条件を満たしたタスクをアイドルプロセッサに割り当てて実行させる処理である。

【0005】例えば、次のようなプログラムを考える。尚、左端の番号はプログラムの行番号である。

```
1:#define  N 1000
2:int a[N],b[N],c[N],i,j,k;
3:main(){
4:      for(i=0;i<N;i++) {      /* タスク1*/
5:           a[i] = i * 2;
6:      }
7:      for(j=1;j<N;j++) {      /* タスク2*/
8:           if(j==1) {b[0] = 0;}
9:           b[j] = a[j] + b[j-1];
10:          if(j==N-1) {printf("b[N-1] = %d¥n",b[j]);}
11:     }
12:     for(k=1;k<N;k++) {      /* タスク3*/
13:          if(k==1) {c[0] = 0;}
14:          c[k] = a[k] + c[k-1];
15:     }
16:}
```

【0006】このプログラム例には３つのループがある。これを１つのループを１つのタスクとしてタスク並列化すると、プログラムの４～６行目がタスク１、７～１１行目がタスク２、１２～１５行目がタスク３になる。タスク間の制御の流れを考慮してタスクにまたがって参照される変数を調べると、タスク１で定義された配列ａがタスク２およびタスク３で使用されているため、タスク２およびタスク３はタスク１の終了後でないと実※行してはいけないことがわかる。ここで、定義とは変数に値を代入すること、使用とは変数の値を用いることである。

【0007】以上より、各タスクの実行可能条件は、タスク１に関しては無条件(いつでも実行開始可能)、タスク２とタスク３に関してはタスク１の終了となる。このタスク並列化プログラムを、プロセッサ２台を用いて並列実行した場合の各タスクの実行の様子を示したのが図

１５のタスク実行グラフである。

【０００８】図１５は、タスク実行状況を表わすタスク実行グラフの一例を示す説明図である。図１５のタスク実行グラフ１５００１において、横軸はプログラム実行開始からの時間、縦軸はプロセッサ番号、グラフ中の四角が各タスクが実行された区間である。

【０００９】実行可能条件と図１５から分かるように、タスク１と同時に実行できるタスクが存在しないため、プロセッサ（０）がタスク１を実行している間はプロセッサ（１）がアイドルプロセッサとなる。

【００１０】

【発明が解決しようとする課題】解決しようとする問題点は、従来の技術では、プログラム実行中のある時点で、実行可能なタスク数が利用可能なプロセッサ数より少ない場合に発生するアイドルプロセッサを有効利用することができない点である。

【００１１】本発明の目的は、これら従来技術の課題を解決し、アイドルプロセッサを有効利用して、実行時間が短縮されたプログラムまたはオブジェクトコードの出力を可能とするタスク並列化方法を提供することである。

【００１２】

【課題を解決するための手段】上記目的を達成するため、本発明のタスク並列化方法は、ソースプログラムを、並列計算機で実行可能な複数のタスクと、該タスクをプロセッサへ割り当てるタスクスケジュール処理とを有するプログラムもしくはオブジェクトコードに変換する並列化コンパイラにおけるタスクの並列化方法であって、（ａ）所定の条件を満たしたタスクに関して、このタスク内で参照される可能性のあるデータや、そのタスクに含まれる命令コードをコンパイル時に検出し、（ｂ）これらのデータや命令コードを、もし、それらが格納されている記憶装置が、タスクスケジュール処理によってこのタスクが割り当てられるプロセッサから見て、最も近い記憶装置なら何もせず、そうでなければ、より近い別の記憶装置へ転送する命令からなる情報転送タスクを生成し、（ｃ）タスクを実行していないアイドルプロセッサがないか監視し、アイドルプロセッサを発見したら、このアイドルプロセッサ以外のプロセッサで実行されているタスクの終了時刻を予測し、予測される終了時刻が最も早いタスクから、アイドルプロセッサに次に割り当てられるタスクである次実行タスクを求め、この次実行タスクに対する情報転送タスクがアイドルプロセッサで実行されるよう割り当てる命令からなる情報転送タスクスケジュール処理を、タスクスケジュール処理に追加する処理を行なうことを特徴とする。

【００１３】

【発明の実施の形態】以下、本発明の実施の形態を、図面により詳細に説明する。図１は、本発明のタスク並列化方法に係る処理動作例を示すフローチャートであり、図２は、図１におけるタスク並列化方法によるタスクの実行状態の概要例を示す説明図、図３は、本発明のタスク並列化方法を行う並列化コンパイラの構成例を示すブロック図、図４は、図３における並列化コンパイラを実行するシステムのハードウェア構成例を示すブロック図である。

【００１４】まず、図２を用いて本発明のタスク並列化方法による処理動作の特徴を説明する。図２において、図２（ａ）は本発明のタスク並列化方法によるタスク１～３の実行状態例を示し、図２（ｂ）は従来技術によるタスク１～３の実行状態を示している。尚、タスク１とタスク２は同時に実行可能であり、タスク３は、タスク１とタスク２の終了後でないと実行できない。

【００１５】すなわち、図２（ｂ）に示すように、従来技術では、タスク２の実行が終了したプロセッサ（２）は、プロセッサ（１）によるタスク１の実行が終了した後、タスク３の実行を開始している。このタスク３の実行には、タスク３で参照されるデータやタスク３に含まれる命令コードの共有メモリから例えばキャッシュへの転送処理が含まれるものとする。

【００１６】そこで、本例のタスク並列化方法では、図２（ａ）に示すように、タスク２の実行が終了したプロセッサ（２）は、プロセッサ（１）によるタスク１の実行中に、タスク３で参照されるデータやタスク３に含まれる命令コードをキャッシュに転送しておく（プリフェッチタスク）。

【００１７】そして、プロセッサ（１）によるタスク１の実行が終了した後、キャッシュにアクセスしてタスク３の実行を開始する。このことにより、タスク３の実行終了時間を、従来技術に比べて、Ｔ時間だけ早めることができる。

【００１８】以下、このようなタスク並列化に関わる並列化コンパイラの処理を、図１を用いて説明する。本例の並列化コンパイラは、ソースプログラムを入力して、並列計算機で実行可能な複数のタスクと、このタスクをプロセッサへ割り当てるタスクスケジュール処理とから構成されるプログラムもしくはオブジェクトコードを出力するものである。

【００１９】そのタスクの並列化方法として、まず、コンパイル時に、実行を開始するための所定の条件（実行可能条件）を満たすタスクに関して、このタスク内で参照される可能性のあるデータ、もしくは、このタスクに含まれる命令コードを検出する（ステップ１０１）。

【００２０】次に、検出したデータまたは命令コードが格納されている記憶装置が、このタスクが上述のタスクスケジュール処理によって割り当てられるプロセッサから見て最も近い記憶装置であるか否かを判別し、最も近い記憶装置であれば何もせず、そうでなければ、より近い別の記憶装置へ転送する命令からなる情報転送タスクを生成する（ステップ１０２）。

【００２１】最後に、タスクを実行していないアイドルプロセッサがないか監視し、アイドルプロセッサを発見したら、このアイドルプロセッサ以外のプロセッサで実行されているタスクの終了時刻を予測し、予測される終了時刻が最も早いタスクから、アイドルプロセッサに次に割り当てられるタスクである次実行タスクを求め、この次実行タスクに対する情報転送タスクが、アイドルプロセッサで実行されるよう割り当てる命令からなる情報転送タスクスケジュール処理を、次実行タスクをアイドルプロセッサへ割り当てるタスクスケジュール処理に追加する（ステップ１０３）。

【００２２】以上の処理により、図２（ａ）に示すように、タスク２の実行が終了したアイドルプロセッサとしてのプロセッサ（２）のタスクスケジュール処理に情報転送タスクスケジュール処理が追加され、その結果、プロセッサ（２）では、プロセッサ（１）によるタスク１の実行中に、タスク３で参照されるデータやタスク３に含まれる命令コードをキャッシュに転送しておく（プリフェッチタスク）ことができる。

【００２３】次に、図４を用いて、このようなタスク並列化を行う並列化コンパイラを実行するシステムのハードウェア構成を説明する。

【００２４】図４において、４１はＣＲＴ(Cathode Ray Tube)等からなり文字や画像を表示出力する表示装置、４２はキーボードやマウス等からなり操作者からの指示を入力する入力装置、４３はＨＤＤ(Hard Disk Drive)等からなり大容量のデータやプログラムを記憶する外部記憶装置、４４はＣＰＵ(Central Processing Unit)や主メモリを有して蓄積プログラム方式による演算処理を行なう情報処理装置、４５は本発明の処理手順に係わるプログラムやデータを記録した記録媒体としての光ディスク、４６は情報処理装置４４からの指示に基づき外部記憶装置４３に記憶させる光ディスク４５内のデータやプログラムを読み出す駆動装置である。

【００２５】情報処理装置４４は、外部記憶装置４３に記憶した光ディスク４５からのデータやプログラムを主メモリにロードすることにより、図３に示す各部からなるタスク並列化コンパイラ１０を構成する。以下、図３を用いてタスク並列化コンパイラ１０を説明する。

【００２６】図３に示すように、タスク並列化コンパイラ１０は、構文解析部１１、タスク並列化部１３、最適化部１５、コード生成部１７から構成され、入力プログラム９０をコンパイルして出力プログラムを生成する。

【００２７】以下、各構成部の説明を行う。構文解析部１１は、入力プログラム９０を入力して中間語９１を出力する。構文解析部１１の処理は通常のコンパイラの場合と特に変わらない。タスク並列化部１３は、中間語９１を入力し、タスク並列化された中間語９１を出力するものであり、以下に説明する依存解析部１３１、タスク解析部１３２、転送情報検出部１３３、中間語変換部１３４から構成されている。

【００２８】依存解析部１３１は、中間語９１を入力してデータ依存関係を解析する。尚、この依存解析部１３１の処理は通常のコンパイラの場合と特に変わらない。タスク解析部１３２は、中間語９１を入力してタスク並列性解析を行なう。このタスク解析部１３２の処理は、本多弘樹、岩田雅彦、笠原博徳　著「Ｆｏｒｔｒａｎプログラム粗粒度タスク間の並列性検出手法」電気情報通信学会論文誌D-I、１９９０年１２月号、Vol.Ｊ７３-Ｄ-I、No.１２、９５１－９６０ページ(以降、「文献２」と記載)で述べられている方法と特に変わらない。

【００２９】転送情報検出部１３３は、中間語９１を入力して、上述したように実行可能条件を満たすタスクに関して、このタスク内で参照される可能性のあるデータまたは、このタスクに含まれる命令コードを検出する等、「情報転送タスク」の生成に必要な情報を解析し、解析結果をタスクテーブル９３と配列参照範囲テーブル９４に出力する。

【００３０】中間語変換部１３４は、中間語９１、タスクテーブル９３、配列参照範囲テーブル９４を入力して、「情報転送タスク」および「情報転送タスクスケジュール処理」を含むタスク並列化された中間語９１を出力するものであり、以下に説明する中間語並列化部１３４１、タスクスケジュール処理生成部１３４２、タスクスケジュール処理拡張部１３４３、情報転送タスク生成部１３４４から構成されている。

【００３１】中間語変換部１３４１は、中間語９１を入力して、タスク並列化された中間語９１を出力する。タスクスケジュール処理生成部１３４２は、中間語変換部１３４１が出力した中間語９１を入力して、タスクスケジュール処理を含むタスク並列化された中間語９１を出力する。

【００３２】タスクスケジュール処理拡張部１３４３は、タスクスケジュール処理生成部１３４２が出力した中間語９１を入力して、「情報転送タスクスケジュール処理」を追加したタスクスケジュール処理を含むタスク並列化された中間語９１を出力する。

【００３３】情報転送タスク生成部１３４４は、タスクスケジュール処理拡張部１３４３が出力した中間語９１を入力して、情報転送タスク、情報転送タスクスケジュール処理を追加したタスクスケジュール処理を含むタスク並列化された中間語９１を出力する。以上の転送情報検出部１３３と中間語変換部１３４の処理は、それぞれ本発明に係わるものである。

【００３４】最適化部１５は、タスク並列化部１３でタスク並列化された中間語９１を入力して最適化された中間語９１を出力する。コード生成部１７は、最適化部１５で最適化された中間語９１を入力して、タスク並列化されたプログラムまたはオブジェクトコードを出力する。

【００３５】これらの最適化部１５とコード生成部１７の処理は、通常のコンパイラの場合と特に変わらない。以下、このような構成のタスク並列化コンパイラ１０によるタスクの並列化動作例を、図５を用いて説明する。

【００３６】図５は、図３におけるタスク並列化コンパイラを実装する並列計算機システムの構成例を示すブロック図である。本図５における並列計算機システム５１は、プロセッサ５１１１～５１１ｎ、キャッシュメモリ５１７１～５１７ｎ、共有メモリ５１５、入出力用プロセッサ５１２、入出力用コンソール５１９、それらを結合する相互結合ネットワーク５１３から構成される。

【００３７】図３のタスク並列化コンパイラ１０は、入出力用コンソール５１９において実行され、これにより、図３の入力プログラム９０が並列ソースプログラムに変換される。さらに、この変換された並列ソースプログラムは、プロセッサ５１１１～５１１ｎ向けコンパイラによって並列オブジェクトプログラムに変換される。

【００３８】そして、この並列オブジェクトプログラムは、リンカによりロードモジュールに変換され、入出力用プロセッサ５１２を通じて共有メモリ５１５にロードされ、各プロセッサ５１１１～５１１ｎにより実行される。

【００３９】この際、共有メモリ５１５にロードされたロードモジュールでは、次実行タスクで参照される配列が存在する記憶装置である共有メモリ５１５に対するアクセスを、情報転送タスクがアイドルプロセッサで予め実行する。

【００４０】そのため、次実行タスクの実行開始時には、この次実行タスクで参照される配列は、共有メモリ５１５よりもプロセッサプロセッサ５１１１～５１１ｎに近い別の記憶装置であるキャッシュメモリ５１７１～５１７ｎに存在する。その結果、次実行タスクの実行時間が短くなるので、プログラム実行時間を短縮することが可能である。

【００４１】以下、図３における構成のタスク並列化コンパイラ１０の具体的な動作例を、図６に示す入力プログラムを用いて説明する。図６は、図３における入力プログラムの一例を示す説明図である。

【００４２】図６の入力プログラム９０における左端にある番号は行番号である。また、１行目は定数Ｎの値を１０００と定義する文、２行目は、整数変数ｉ，ｊ，ｋ，ｍと、第２次元目が０～Ｎ－１の添字を持つ整数型の１次元配列ａ，ｂ，ｃの宣言文である。

【００４３】３～２０行目は入力プログラム９０の主処理関数mainであり、４～６行目はiをループ制御変数とするループである。以下、ループは先頭行の行番号を用いて表す。すなわち、このループはループ４と表す。

【００４４】７～１１行目はｊをループ制御変数とするループ７、１２～１５行目はｋをループ制御変数とするループ１２、１６～１９行目はｍをループ制御変数とするループ１６である。

【００４５】図７および図８は、図１における出力プログラムの一例を示す説明図である。図７および図８の左端にある番号は行番号である。

【００４６】１行目は定数Ｎの値を１０００と定義する文、２行目は、整数変数ｉ，ｊ，ｋ，ｍと、第１次元目が０～Ｎ－１の添字範囲を持つ整数型の１次元配列ａ，ｂ，ｃの宣言文、３～６行目は、定数INIT_EXEC_MT_NUM、NPE、NTASK、NO_TASKの値を、それぞれ「－９９」、「２」、「４」、「－９８」に定義する文である。

【００４７】７行目は整数変数newMT,succMT,tmp5,tmp6,tmp7,tmp8,0～NPE-1の添字範囲を持つ整数型の１次元配列ExecMTの宣言文、８行目は整数変数ii,kk,kk,mm,myPEの宣言文、９～１５行目は、複素数変数TaskGranularity、整数変数SuccTaskNo、複素数変数StartTime、真偽値型変数Finishを要素として持つ構造体TaskDataの宣言文、１６行目は、０～NTASKの添字範囲を持ち、構造体TaskDataを要素とする配列TDataの宣言文である。

【００４８】１７～８９行目が出力プログラム９２の主処理関数mainである。そのうち、２７～３２、３７～３８、４０～４１、４５～４６、５２～５３、５８～５９、６４、８４、８６～８７行目がタスクスケジュール処理を行なう部分である。

【００４９】４２～４４行目がタスク１、４７～５１行目がタスク２、５４～５７行目がタスク３、６０～６３行目がタスク４、１８～２６、３３～３６、３９、６５～６７、７２～７３、７８～７９、８３、８５行目が情報転送タスクスケジュール処理を行なう部分、６８～７１行目がタスク２に対する情報転送タスク、７４～７７行目がタスク３に対する情報転送タスク、８０～８２行目がタスク４に対する情報転送タスクである。

【００５０】以下、このような図６に示す入力プログラム９０と図７，８に示す出力プログラム９２に関わる図３のタスク並列化コンパイラ１０内の個々の処理を説明する。

【００５１】まず、構文解析部１１により、入力プログラム９０を入力して中間語９１を出力する。尚、中間語９１は図６の入力プログラム９０に対応しているので、以下の説明では、図６の入力プログラム９０を中間語９１のソースプログラムイメージの表現として用いる。

【００５２】次に、タスク並列化部１３は、依存解析部１３１、タスク解析部１３２、転送情報検出部１３３、中間語変換部１３４により次のような処理を行う。

【００５３】依存解析部１３１では、Alfred V.Aho、Ravi Sethi、Jeffrey D.Ullman著、「Compilers」、Addison-Wesley Publishing Company、1986に説明されている処理により、中間語９１を入力してデータ依存関係を解析する。

【００５４】タスク解析部１３２は、中間語９１を入力してタスク並列性解析を行なう。図６の入力プログラム

９０では、ループ４がタスク１、ループ７がタスク２、ループ１２がタスク３、ループ１６がタスク４と解析される。また、タスク１～４の実行可能条件は、それぞれ「タスク１：条件なし」、「タスク２：タスク１の終了」、「タスク３：タスク１の終了」、「タスク４：タスク３の終了」と解析される。

【００５５】これらの実行可能条件をまとめて図９に示すように記憶する。図９は、図３のタスク解析部により図６の入力プログラムから解析した実行可能条件をまとめた表の構成例を示す説明図である。

【００５６】本例の実行可能条件の表９５においては、各タスク１～４の実行可能条件が、「タスク１：条件なし」、「タスク２：タスク１の終了」、「タスク３：タスク１の終了」、「タスク４：タスク３の終了」としてまとめて登録されている。

【００５７】さらに、タスク解析部１３２は、そのタスクの実行終了をプログラムの終了とみなせる唯一のタスクである「プログラム終了タスク」を、以下の一般的な技術により求める。

【００５８】すなわち、タスクをノード、タスク間の制御依存関係をエッジとする「タスクグラフ」を作成し、この「タスクグラフ」に処理を含まないダミータスクである「プログラム終了タスク」を加え、エッジの始点を持たないタスクから「プログラム終了タスク」へのエッジを設けて、「プログラム終了タスク」とする。

【００５９】但し、「プログラム終了タスク」を終点とするエッジが１本しかない場合は、そのエッジの始点であるタスクを「プログラム終了タスク」とみなせるので、図６の入力プログラム９０では、タスク４をプログラム終了タスクとすることができる。

【００６０】このようなタスク解析部１３２の処理の詳細に関しては、上述の文献２で述べられている技術と特に変わらないので、これ以上は述べない。

【００６１】次に、転送情報検出部１３３は、中間語９１を入力して、タスク１～４の各々に対し、タスク内で参照される配列の参照範囲を解析してその結果を配列参照範囲テーブル９４に出力し、タスク実行時間の見積もりと、そのタスクの終了で初めて実行可能条件が満たされるタスク総数の解析を行ない、それらの結果をタスクテーブル９３に出力する。

【００６２】ここで、参照とは変数が定義または使用されること、変数とはスカラ変数または配列のこと、定義とは変数に値を代入すること、使用とは変数の値を用いること、配列の参照範囲とは、その配列の参照される可能性がある添字範囲のことを指す。

【００６３】以下、転送情報検出部１３３の処理動作例を、図１０と図１１を用いて説明する。図１０は、図３における転送情報検出部の処理手順例を示すフローチャートであり、図１１は、図１０における転送情報検出部の処理手順により得られるタスクテーブルと配列参照範囲テーブルの構成例を示す説明図である。

【００６４】図３の転送情報検出部１３３が図６の入力プログラム９０に対して、図１０におけるステップ１３３１～１３３６の処理を行うことにより、図１１に示すタスクテーブル９３１～９３４と配列参照範囲テーブル９４２～９４６が得られる。

【００６５】図１１においては、タスクテーブル９３２と配列参照範囲テーブル９４２，９４３のみフィールドを詳細に示している。タスクテーブル９３１～９３４は、それぞれ図６のタスク１～４に対応するタスクテーブルである。

【００６６】以下、タスクテーブル９３２を例にとって、タスクテーブルの各フィールド９３２１～９３２５を説明する。フィールド９３２１には次のタスクテーブルへのポインタを格納する。図中、フィールド９３２１から右に向いた矢印がこのポインタに対応する。次のタスクテーブルがない場合はNULL値を格納する。

【００６７】フィールド９３２２にはタスク番号を格納する(図中、「2」が記載)。フィールド９３２３には、そのタスクに関して見積もられたタスク実行時間を格納する(図中、「６００５」が記載)。

【００６８】フィールド９３２４には、そのタスクの終了で初めて実行可能条件が満たされるタスクの総数である次実行タスク数を格納する(図中、「0」が記載)。フィールド９３２５には、そのタスク内で参照される配列の配列参照範囲テーブルへのポインタを格納する。図中、フィールド９３２５から下に向いた矢印がこのポインタに対応する。そのタスクに関して配列参照範囲テーブルがない場合はNULL値を格納する。

【００６９】次に、配列参照範囲テーブル９４２～９４６において、図６のタスク２の解析結果を収めたのが配列参照範囲テーブル９４２と９４３、タスク３の解析結果を収めたのが配列参照範囲テーブル９４４と９４５、タスク４の解析結果を収めたのが配列参照範囲テーブル９４６である。タスク１は配列参照範囲テーブルをもたない。

【００７０】以下、配列参照範囲テーブル９４２を例にとって、配列参照範囲テーブルのフィールド９４２１～９４２３を説明する。フィールド９４２１には、そのタスク内で参照される配列の配列名を格納する（図中、「ａ」が記載）。

【００７１】フィールド９４２２には、そのタスク内で参照される配列の参照範囲を「配列添字の下限値：配列添字の上限値」の形式で格納する（図中、「１：Ｎ－１」が記載）。

【００７２】フィールド９４２３には、そのタスク内で参照される次の配列の配列参照範囲テーブルへのポインタを格納する。図中、フィールド９４２３から下に向いた矢印がこのポインタに対応する。次の配列参照範囲テーブルがない場合はNULL値を格納する。

【００７３】以下、このような転送情報検出部１３３の処理を中間語９１に適用した結果について説明する。まず、図１０のステップ１３３１の処理を適用する。ここでは、転送情報検出部１３３で未処理のタスクの例としてタスク２を選択する。

【００７４】次に、ステップ１３３２の処理を中間語９１に適用する。ステップ１３３２は、所定の条件を満たすタスクを選択する処理である。タスク解析部１３２で解析したタスク２の実行可能条件は「タスク１の終了」なので常に真ではない。従って、NO方向へ処理が分岐し、ステップ１３３３へ移る。

【００７５】次に、ステップ１３３３の処理を中間語９１に適用すると、タスク２内で参照される配列ａ、配列ｂの配列参照範囲が次のように解析される。まず、タスク２のループ制御変数jが１～Ｎ－１の値をとることから、添字がjである９行目の配列ａの参照範囲は「１：Ｎ－１」と解析される。

【００７６】同様に配列ｂに関しては、添字が０である８行目での参照範囲は「０」、添字がjである９行目左辺での参照範囲は「１：Ｎ－１」、添字がj-1である９行目右辺での参照範囲は「０：Ｎ－２」、添字がN-1である１０行目での参照範囲は「Ｎ－１」と解析されるので、これらの和集合をとって、タスク２での配列ｂの参照範囲が「０：Ｎ－１」となる。

【００７７】以上の解析結果は配列参照テーブル９４２、９４３に格納される。まず、配列名ａが配列参照テーブル９４２のフィールド９４２１に、配列参照範囲「１：Ｎ－１」がフィールド９４２２に、配列名ｂが配列参照テーブル９４３のフィールド９４３１に、配列参照範囲「０：Ｎ－１」がフィールド９４３２に、それぞれ格納される。

【００７８】また、配列参照範囲テーブル９４２へのポインタがタスクテーブル９３２のフィールド９３２５に、配列参照範囲テーブル９４３へのポインタが配列参照テーブル９４２のフィールド９４２３に、配列参照テーブル９４３のフィールド９４３３にNULLが、それぞれ格納される。

【００７９】次に、ステップ１３３４を中間語９１に適用すると、タスク２のタスク実行時間であるコストが以下のようにして見積もられる。まず、図６の入力プログラム９０の１行目よりNは１０００なので、入力プログラム９０タスク２の７行目のループは９９９回まわることがわかる。

【００８０】タスク２の８行目では、if文の条件判定をコスト１、帰結節の代入文b[0]=0をコスト１と見積もる。この条件判定は各ループ繰り返しで、帰結節はループ制御変数jが１の時のみ実行されるから、８行目の合計コストは１０００となる。

【００８１】タスク２の９行目では、代入文右辺のa[j]およびb[j-1]のロード、加算、左辺のb[j]のストアを各々コスト１と見積もる。この代入文は各ループ繰り返しで実行されるから、９行目の合計コストは３９９６となる。

【００８２】タスク２の１０行目では、if文の条件判定をコスト１、帰結節のprintf文をコスト１０と見積もる。この条件判定は各ループ繰り返しで、帰結節はループ制御変数jが９９９の時のみ実行されるから、１０行目の合計コストは１０９９となる。

【００８３】以上を合計すると、タスク２のコストは６００５となり、この値が図１１におけるタスクテーブル９３１のフィールド９３１３に格納される。

【００８４】次に、ステップ１３３５を中間語９１に適用する。図３のタスク解析部１３２で解析された実行可能条件を表にした図９より、「タスク２の終了」を実行可能条件にもつタスクは存在しないことから、タスク２の終了で初めて実行可能条件が満たされるタスクの総数は０となり、その値が図１１におけるタスクテーブル９３２のフィールド９３２４に格納される。

【００８５】次に、ステップ１３３６を中間語９１に適用する。タスク１，３，４のうち転送情報検出部１３３で未処理のものが存在すれば、NO方向へ処理が分岐してステップ１３３１へ戻り、次のタスクに関してステップ１３３２～１３３６を繰り返し、存在しなければ転送情報検出部１３３を終了する。以上で、転送情報検出部１３３の処理動作例の説明を終了する。

【００８６】次に、図３の中間語変換部１３４の処理を図６の入力プログラム９０に適用した結果について説明する。中間語変換部１３４は、中間語９１、タスクテーブル９３、配列参照範囲テーブル９４を入力して、タスクスケジュール処理、情報転送タスクスケジュール処理、情報転送タスクを持つタスク並列化された中間語９１を出力する。

【００８７】尚、タスク並列化された中間語９１は図７，８の出力プログラム９２に対応しているので、以下の説明では、図７，８の出力プログラム９２をタスク並列化された中間語９１のソースプログラムイメージの表現として用いる。

【００８８】まず、図３の中間語変換部１３４は、中間語並列化部１３４１の処理を中間語９１に適用する。この結果挿入された中間語が、図７，８の出力プログラム９２の３～６、２８～２９、８８行目である。

【００８９】３～６行目は変数を定義する文である。２８行目は並列実行部分の開始を表すコンパイラ指示文であり、# pragma omp parallelで並列実行部分の開始を示し、PRIVATE(myPE,newMT)で、プロセッサ番号を表す変数myPE、変数newMTが各プロセッサで別々の変数になるように指示している。

【００９０】８８行目は並列実行部分の終了を示すコンパイラ指示文である。２９行目はプロセッサ番号を表す変数myPEの設定文であり、この文の右辺はプロセッサ番

号問い合わせ関数get_processor_numである。

【0091】次に、タスクスケジュール処理生成部134の処理を中間語91に適用する。この結果挿入されたタスクスケジュール処理にあたる中間語が、図7，8の出力プログラム92の30～32行目、37行目、40～41行目、45～46行目、52～53行目、58～59行目、64行目、84～85行目、87行目である。

【0092】30、87行目のwhileループは、プログラム終了タスクであるタスク4が実行終了するまで全プロセッサがwhileループを回ってタスクスケジュール処理を実行し続ける処理である。

【0093】31、37行目は、この2文の間がクリティカルセクションであることを示す指示文であり、この2つの指示文で挟まれた部分は、1度に1台のプロセッサでしか実行できない排他処理であることを表す。

【0094】32行目の代入文では、実行可能条件を満たしたタスクのタスク番号を関数GET_MT_FROM_QUEUEで取り出し、変数newMTに設定している。実行可能条件を満たしたタスクが存在しない場合は、該関数は定数変数NO_TASKを返す。この関数の処理の内容は、上述の文献1で述べられている技術と特に変わらないので、ここでは詳細には述べない。

【0095】40～41行目、45～46行目、52～53行目、58～59行目、64行目、84行目は、32行目で変数newMTに設定されたタスク番号に従って実行するタスクを選択する部分である。85行目は、実行終了したタスクの終了フラグを設定する処理である。終了フラグはタスク実行終了を示すフラグであり、タスク毎に設けられている。

【0096】次に、タスクスケジュール処理拡張部143の処理を中間語91に適用した場合を説明する。図12は、図3におけるタスクスケジュール処理拡張部の処理手順例を示すフローチャートである。

【0097】図12に示すように、図3におけるタスクスケジュール処理拡張部1343は、ステップ13431～13436の各処理を、中間語に対して行う。

【0098】まず、ステップ13431の処理を中間語91に適用する。この結果挿入された中間語が、図7，8の出力プログラム92における65～67行目、72～73行目、78～79行目、83行目である。

【0099】ここで挿入された中間語は、変数newMTに情報転送タスクのタスク番号が設定されていたら、その情報転送タスクを実行する処理を意味する。ここでは、情報転送タスクのタスク番号は、情報転送タスクに対する元のタスクのタスク番号に、タスク総数を加えたものとする。図7，8の出力プログラム92では、タスク1～4に対する情報転送タスクには、それぞれ5～8のタスク番号を与える。

【0100】次に、ステップ13432の処理を中間語91に適用する。この結果挿入された中間語が、図7，8の出力プログラム92の39行目であり、情報転送タスクを除くタスクの実行開始時刻を構造体TDataに設定する文である。この文の右辺の関数present_timeは、現時刻を与える関数である。

【0101】次に、ステップ13433の処理を中間語91に適用する。この結果挿入された中間語が、図7，8の出力プログラム92の27行目、38行目、86行目である。27行目は配列ExecMTの初期化文、38行目は各プロセッサが現在実行しているタスクのタスク番号を配列ExecMTに設定する文、86行目は配列ExecMTの値を初期値に戻す文である。

【0102】次に、ステップ13434の処理を中間語91に適用する。この結果挿入された中間語が、図7，8の出力プログラム92の33、36行目である。これは、関数GET_MT_FROM_QUEUEが返した値が、その時点で実行可能条件を満たしたタスクが存在しないことを意味する定数変数NO_TASKかどうかを調べる文である。

【0103】次に、ステップ13435の処理を中間語91に適用する。この結果挿入された中間語が、図7，8の出力プログラム92の18～26行目、34行目の文である。

【0104】18～25行目では、図11に示すタスクテーブル931～934を用いて、転送情報検出部133のステップ1334で見積もられたタスク実行時間を構造体TDataのTaskGranularityフィールドに、図10のステップ1335で数えられた次実行タスク数を構造体TDataのSuccTaskNoフィールドに設定する。

【0105】例えばタスク2の場合では、図11におけるタスクテーブル932のフィールド9322に格納されている値2を、図7，8の出力プログラム92の19、23行目の代入文左辺の構造体TDataの添字に、また、フィールド9323に格納されている値「6005」を、出力プログラム92の19行目の代入文右辺に、フィールド9324に格納されいている値0を、出力プログラム92の23行目の代入文右辺に設定する。

【0106】図7，8の出力プログラム92の26行目は各タスクの終了フラグの初期化文である。また、同34行目は取得した次実行タスクのタスク番号を変数newMTに設定する代入文であり、この代入文右辺の関数PredictSuccMTは、次実行タスクの番号を取得するための関数（次実行タスク番号取得関数）である。尚、この実行タスク番号取得関数の処理の内容については、後述の図14を用いて説明する。

【0107】最後に、ステップ13436の処理を中間語91に適用する。この結果挿入された中間語が、図7，8の出力プログラム92の35行目である。これは、変数succMTの値である取得された次実行タスクのタスク番号にタスク総数を加えて、次実行タスクに対する情報転送タスクのタスク番号を得る処理である。

【0108】以上で、タスクスケジュール処理拡張部１３４３の処理の説明を終る。次に、情報転送タスク生成部１３４４の処理を中間語９１に適用した場合を説明する。

【0109】図１３は、図３における情報転送タスク生成部の処理手順例を示すフローチャートである。本図１３に示すように、図３における情報転送タスク生成部１３４４はステップ１３４４１～１３４４４からなる処理をおこなう。

【0110】まず、ステップ１３４４１の処理を図３の中間語９１に適用する。ここでは、未処理のタスクの例としてタスク２を選択する。次に、ステップ１３４４２の処理を中間語９１に適用する。図３のタスク解析部１３２で解析したタスク２の実行可能条件は「タスク１の終了」なので常に真ではない。従って、ここでは、NO方向へ処理が分岐してステップ１３３４３へ移る。

【0111】次に、ステップ１３４４３の処理を中間語９１に適用する。図１１で示すタスク２の配列参照範囲テーブル９４２、９４３の情報を利用して挿入された中間語が、図７，８の出力プログラム９２の６８～７１行目である。

【0112】これは、配列参照範囲テーブル９４２のフィールド９４２１に格納されている配列名ａとフィールド９４２２に格納されている参照範囲１：Ｎ－１、配列参照範囲テーブル９４３のフィールド９４３１に格納されている配列名ｂとフィールド９４３２に格納されている参照範囲０：Ｎ－１より作成された、添字１～Ｎ－１の範囲の配列ａの要素と、添字０～Ｎ－１の範囲の配列ｂの要素を使用するループである。

【0113】次に、ステップ１３４４４の処理を中間語９１に適用する。タスク１、３、４のうち、未処理のものが存在すれば、NO方向へ処理が分岐してステップ１３４４１へ戻り、次のタスクに関して同様にステップ１３４４２～１３４４３を繰り返す。また、未処理のタスクが存在しなければステップ１３４４を終了する。

【0114】以上で、図３における情報転送タスク生成部１３４４の処理動作例、および、図３における中間語変換部１３４の処理動作例の説明を終る。

【0115】このようにして、中間語変換部１３４でタスク並列化された中間語９１を、最適化部１５は入力し、最適化された中間語９１を出力する。そして、コード生成部１７は、最適化部１５で最適化された中間語９１を入力して、図７，８に示す出力プログラム９２を出力する。尚、これらの最適化部１５、コード生成部１７の処理の内容は通常のコンパイラの場合と特に変わらないので、ここでは詳細には述べない。

【0116】以上で、本発明に係るタスク並列化方法の一例の説明を終り、図１４で、次実行タスク番号取得関数、すなわち、図７，８の出力プログラム９２の３４行目の代入文右辺の関数PredictSuccMTの処理の内容を説明する。

【0117】図１４は、次実行タスク番号取得関数の処理例を示すフローチャートである。図７，８の出力プログラム９２の３４行目の代入文右辺の関数PredictSuccMTは引数を２個持つ。第１引数は、各タスクのタスク実行時間、次実行タスク数、タスク実行開始時間、タスク終了フラグを格納した構造体TData、第２引数は、各プロセッサで現在実行中のタスク番号を格納した配列ExecMTである。

【0118】本図１４に示すように、次実行タスク番号取得関数は、ステップ２１１～２１４の処理を行う。まずステップ２１１では、現在実行中のタスクを検出する。これは、関数PredictSuccMTの第２引数である配列ExecMTの添字をプロセッサ番号とした要素の値を調べて、実行中のタスクのタスク番号を取得する処理である。

【0119】次にステップ２１２では、実行中の各タスクの実行終了までの残り時間を予測する。これは、関数PredictSuccMTの第１引数である構造体TDataを用いて計算する。現在実行中のタスクのタスク番号をｉとすると、残り時間の予測式は、次のようにして与えられる。

【0120】残り時間
＝見積もられたタスク実行時間－（現時刻－タスク実行開始時刻）
＝TData[i].TaskGranularity－（present_time()－TData[i].StartTime）

【0121】さらにステップ２１３では、ステップ２１２の結果より、残り時間が最小である最小残り時間タスクを求める。そしてステップ２１４では、この最小残り時間タスクの実行終了後に実行可能になるタスクを探し、このタスクの次実行タスク数が最多のタスクを次に実行される次実行タスクとする。

【0122】これは、現在未実行のタスクから、該最小残り時間タスクが終了すると実行可能条件が真になるタスク群を探し、そのタスク群から、TData構造体のSuccTaskNoフィールドの値が最大のタスクを求める処理である。以上で、次実行タスク番号取得関数の説明を終る。

【0123】以上、図１～図１４を用いて説明したように、本例のタスク並列化方法では、タスクを実行していないアイドルプロセッサがないか監視し、このアイドルプロセッサを発見したら、他のプロセッサで実行されているタスクの終了時刻を予測し、予測される終了時刻が最も早いタスクから、アイドルプロセッサに次に割り当てるタスクである次実行タスクを求める。

【0124】そして、この次実行タスクで参照される可能性のあるデータや、そのタスクに含まれる命令コードを、そのアイドルプロセッサのキャッシュに転送する情報転送タスクを作成し、かつ、その情報転送タスクがアイドルプロセッサで実行されるように割り当てる命令からなる情報転送タスクスケジュール処理を作成して、並

列化コンパイラが生成したタスクスケジュール処理に追加する。

【0125】これにより、プロセッサのアイドルタイムにデータをキャッシュに転送するので、アイドルプロセッサとキャッシュの有効利用を図ることができ、プログラム実行中のある時点で実行可能なタスク数が、利用可能なプロセッサ数より少ない場合におけるプログラムまたはオブジェクトコードの実行時間を短縮することが可能である。

【0126】尚、本例のタスク並列化方法により予めキャッシュに読み込んだデータが無効化された場合には、従来技術の通りに共有メモリにアクセスする。

【0127】また、本発明は、図1～図14を用いて説明した例に限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能である。例えば、本例では、アイドルプロセッサに次に割り当てられるタスクで参照される可能性のあるデータや、そのタスクに含まれる命令コードを、そのアイドルプロセッサのキャッシュに転送する情報転送タスクと、その情報転送タスクをアイドルプロセッサに割り当てるスケジュール処理を作成しているが、情報転送タスクとしては、それらのデータや命令コードを外部記憶装置からメインメモリへ、あるいは、他プロセッサのリモートメモリからアイドルプロセッサのローカルメモリに転送するものであっても良い。

【0128】

【発明の効果】本発明によれば、プログラム実行中のある時点で、実行可能なタスク数が利用可能なプロセッサ数より少ない場合でも、タスクが割り当てられていないアイドルプロセッサを有効利用して実行時間を短縮できるプログラムまたはオブジェクトコードを出力することができ、並列計算機システムの性能の向上を図ることが可能である。

【図面の簡単な説明】

【図1】本発明のタスク並列化方法に係る処理動作例を示すフローチャートである。

【図2】図1におけるタスク並列化方法によるタスクの実行状態の概要例を示す説明図である。

【図3】本発明のタスク並列化方法を行う並列化コンパイラの構成例を示すブロック図である。

【図4】図3における並列化コンパイラを実行するシステムのハードウェア構成例を示すブロック図である。

【図5】図3におけるタスク並列化コンパイラを実装する並列計算機システムの構成例を示すブロック図である。

【図6】図3における入力プログラムの一例を示す説明図である。

【図7】図1における出力プログラムの一例の前半部分を示す説明図である。

【図8】図1における出力プログラムの一例の後半部分を示す説明図である。

【図9】図3のタスク解析部により図6の入力プログラムから解析した実行可能条件をまとめた表の構成例を示す説明図である。

【図10】図3における転送情報検出部の処理手順例を示すフローチャートである。

【図11】図10における転送情報検出部の処理手順により得られるタスクテーブルと配列参照範囲テーブルの構成例を示す説明図である。

【図12】図3におけるタスクスケジュール処理拡張部の処理手順例を示すフローチャートである。

【図13】図3における情報転送タスク生成部の処理手順例を示すフローチャートである。

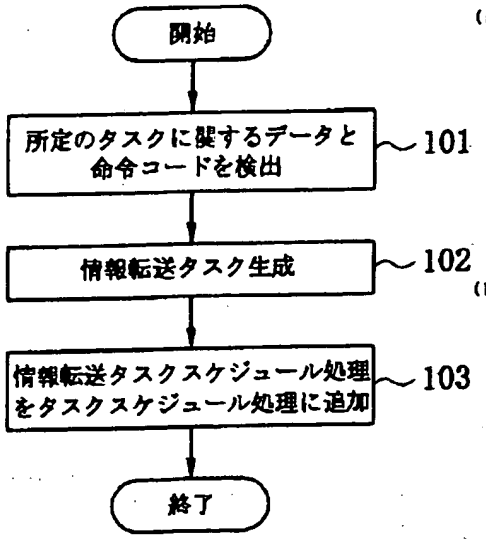
【図14】次実行タスク番号取得関数の処理例を示すフローチャートである。

【図15】タスク実行状況を表わすタスク実行グラフの一例を示す説明図である。

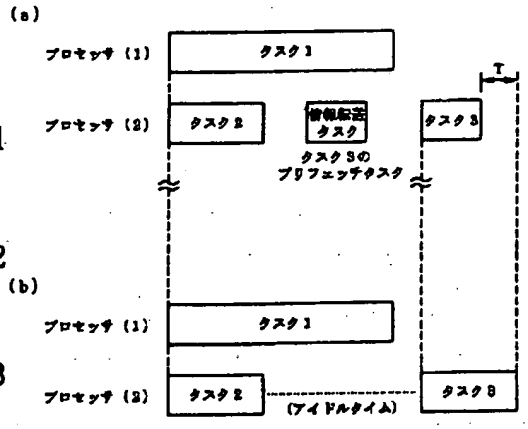
【符号の説明】

10：タスク並列化コンパイラ、11：構文解析部、13：タスク並列化部、15：最適化部、17：コード生成部、131：依存解析部、132：タスク解析部、133：転送情報検出部、134：中間語変換部、1341：中間語並列化部、1342：タスクスケジュール処理生成部、1343：タスクスケジュール処理拡張部、1344：情報転送タスク生成部、41：表示装置、42：入力装置、43：外部記憶装置、44：情報処理装置、45：光ディスク、46：駆動装置、51：並列計算機システム、5111～511n：プロセッサ、512：入出力用プロセッサ、513：相互結合ネットワーク、515：共有メモリ、5171～517n：キャッシュメモリ、519：入出力用コンソール、90：入力プログラム、91：中間語、92：出力プログラム、93，931～934：タスクテーブル、9321～9325：フィールド（タスクテーブル）、94，942～946：配列参照範囲テーブル、9421～9423：フィールド（配列参照範囲テーブル）、95：実行可能条件の表、15001：タスク実行グラフ。

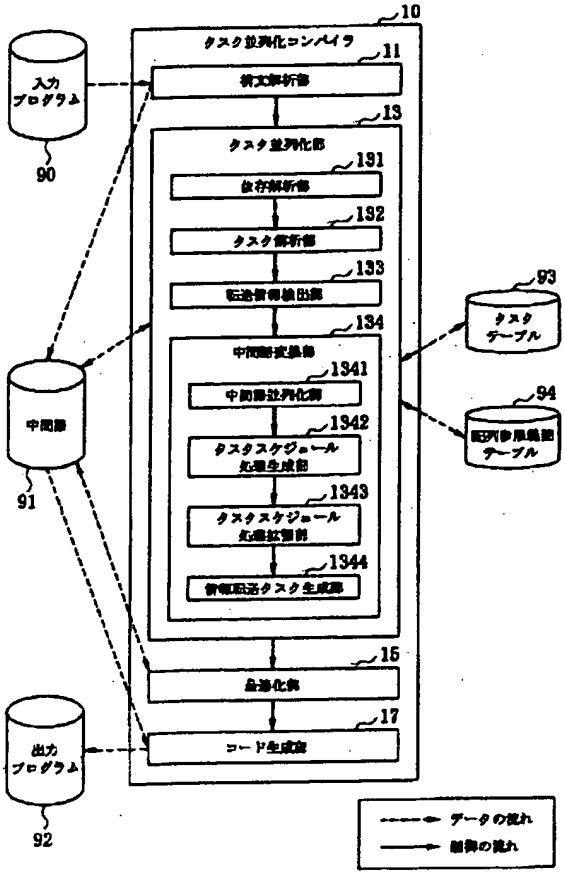
【図1】


開始
所定のタスクに関するデータと命令コードを検出
101
情報転送タスク生成
102
情報転送タスクスケジュール処理をタスクスケジュール処理に追加
103
終了

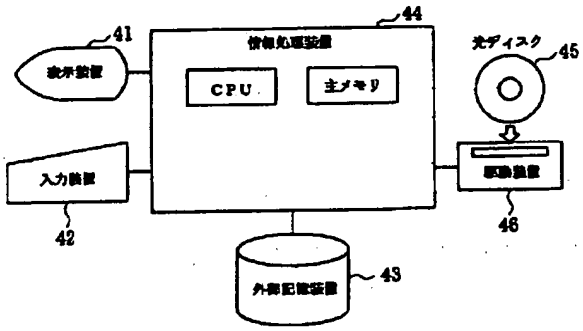
【図2】


(a)
プロセッサ (1)
タスク1
プロセッサ (2)
タスク2
情報転送タスク
タスク3
T
タスク3のプリフェッチタスク
(b)
プロセッサ (1)
タスク1
プロセッサ (2)
タスク2
(アイドルタイム)
タスク3

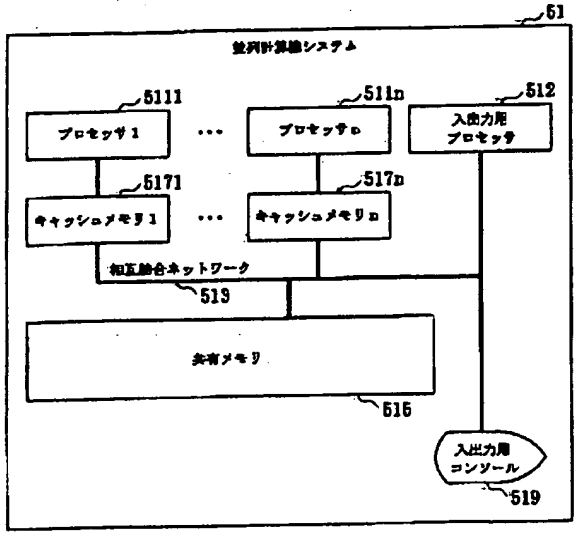
【図3】


10
タスク並列化コンパイラ
11
構文解析部
13
タスク並列化部
131
依存解析部
132
タスク解析部
133
転送情報抽出部
134
中間語変換部
1341
中間語並列化部
1342
タスクスケジュール処理生成部
1343
タスクスケジュール処理拡張部
1344
情報転送タスク生成部
15
最適化部
17
コード生成部
入力プログラム
90
中間語
91
出力プログラム
92
93
タスクテーブル
94
並列計算機構テーブル
データの流れ
制御の流れ

【図4】


41
表示装置
44
情報処理装置
CPU
主メモリ
光ディスク
45
入力装置
42
駆動装置
46
外部記憶装置
43

【図5】


51
並列計算機システム
5111
プロセッサ1
511n
プロセッサn
512
入出力用プロセッサ
5171
キャッシュメモリ1
517n
キャッシュメモリn
相互結合ネットワーク
513
共有メモリ
515
入出力用コンソール
519

【図6】

```
1:#define N 1000
2:int a[N],b[N],c[N],i,j,k,m;
3:main(){
4:   for(i=0;i<N;i++) {
5:      a[i] = i * 2;
6:   }
7:   for(j=1;j<N;j++) {
8:      if(j==1) {b[0] = 0;}
9:      b[j] = a[j] + b[j-1];
10:     if(j==N-1) {printf("b[N-1] = %d¥n",b[N-1]);}
11:  }
12:  for(k=1;k<N;k++) {
13:     if(k==1) {c[0] = 0;}
14:     c[k] = a[k] + c[k-1];
15:  }
16:  for(m=N-1;m=>0;m--) {
17:     c[m] = c[m+1];
18:     if(m==0) {printf("c[0] = %d¥n",c[0]);}
19:  }
20:}
```

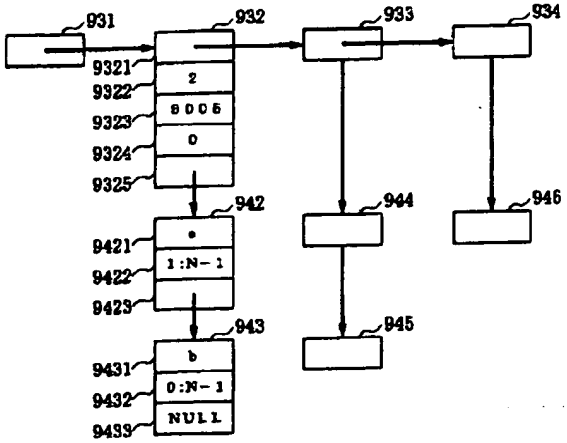
90

【図9】

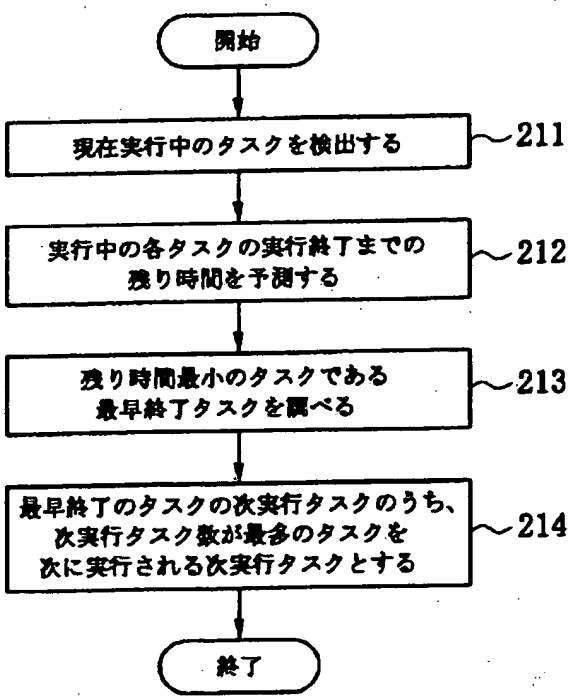
95

タスク番号	実行可能条件
1	条件なし
2	タスク1の終了
3	タスク1の終了
4	タスク3の終了

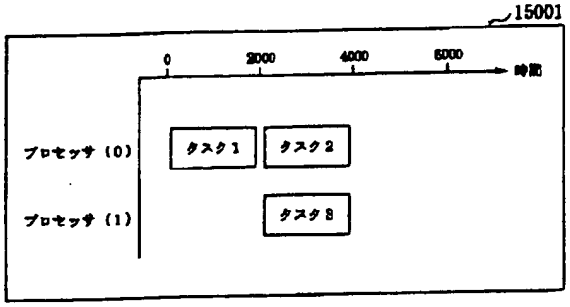
【図11】



【図14】



【図15】



【図7】

```
 1:#define N 1000
 2:int a[N],b[N],c[N],i,j,k,m;
 3:#define INIT_EXEC_MT_NUM -99
 4:#define NPE 2
 5:#define NTASK 4
 6:#define NO_TASK -98
 7:int newMT,succMT,tmp6,tmp7,tmp8, ExecMT[0:NPE-1];
 8:int ii,jj,kk,mm,myPE;
 9:typedef struct TaskData {
10:   double TaskGranularity;
11:   int    SuccTaskNo;
13:   double StartTime;
14:   Boolean Finish;
15:} TaskData;
16:TaskData TData[NTASK+1];
17:main(){
18:   TData[1].TaskGranularity = 2000;
19:   TData[2].TaskGranularity = 6005;
20:   TData[3].TaskGranularity = 4996;
21:   TData[4].TaskGranularity = 3007;
22:   TData[1].SuccTaskNo = 3;
23:   TData[2].SuccTaskNo = 0;
24:   TData[3].SuccTaskNo = 4;
25:   TData[4].SuccTaskNo = 0;
26:   TData[1:4].Finish = FALSE;
27:   ExecMT[0:NPE-1] = INIT_EXEC_MT_NUM;
28:#pragma omp parallel PRIVATE(myPE,newMT)
29:   myPE = get_processor_num();
30:   while(TData[4].Finish){
31:#pragma omp critical section(flag_queue)
32:      newMT = GET_MT_FROM_QUEUE();
33:      if(newMT == NO_TASK) {
34:         succMT = PredictSuccMT(TData,ExecMT);
35:         newMT = succMT + NTASK;
36:      }
37:#pragma omp end critical section(flag_queue)
38:      ExecMT[myPE] = newMT;
39:      if(1<=newMT&&newMT<=NTASK)TData[newMT].StartTime=present_time();
40:      switch(newMT) {
41:         case 1:
42:            for(i=0;i<N;i++) {
43:              a[i] = i * 2;
44:            }
45:            break;
```

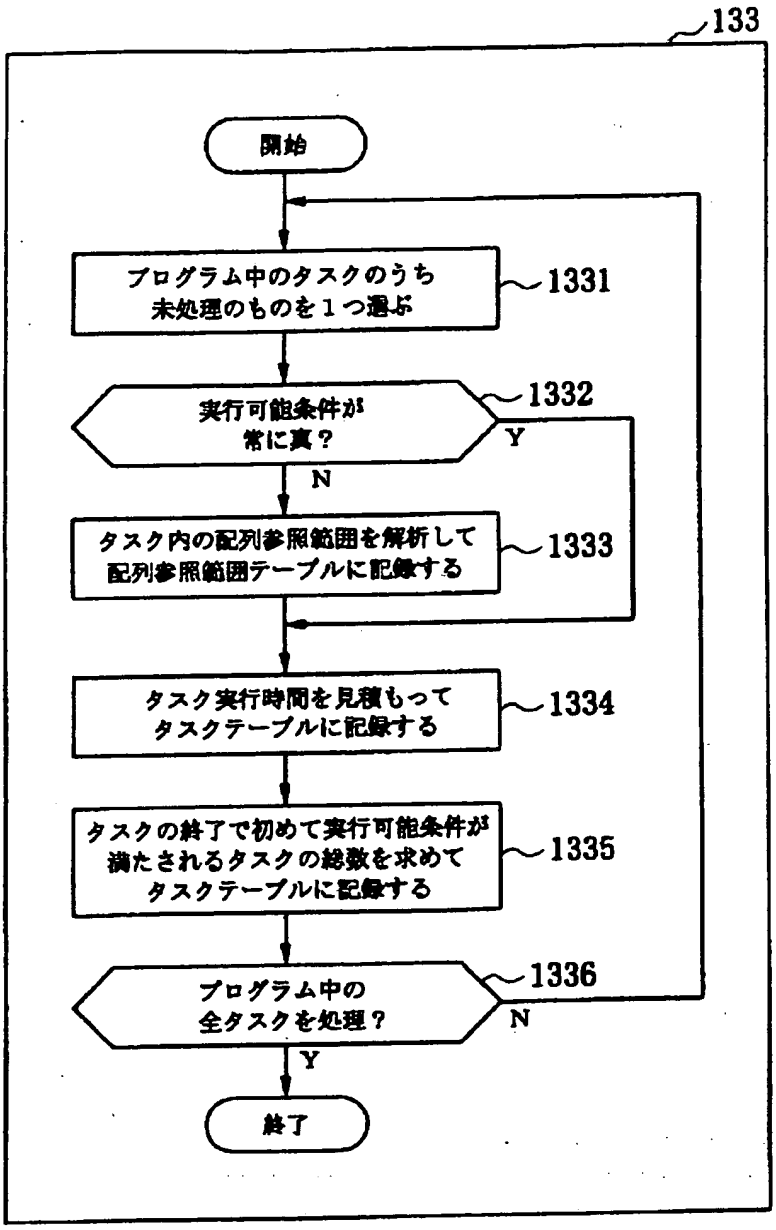
92

【図8】

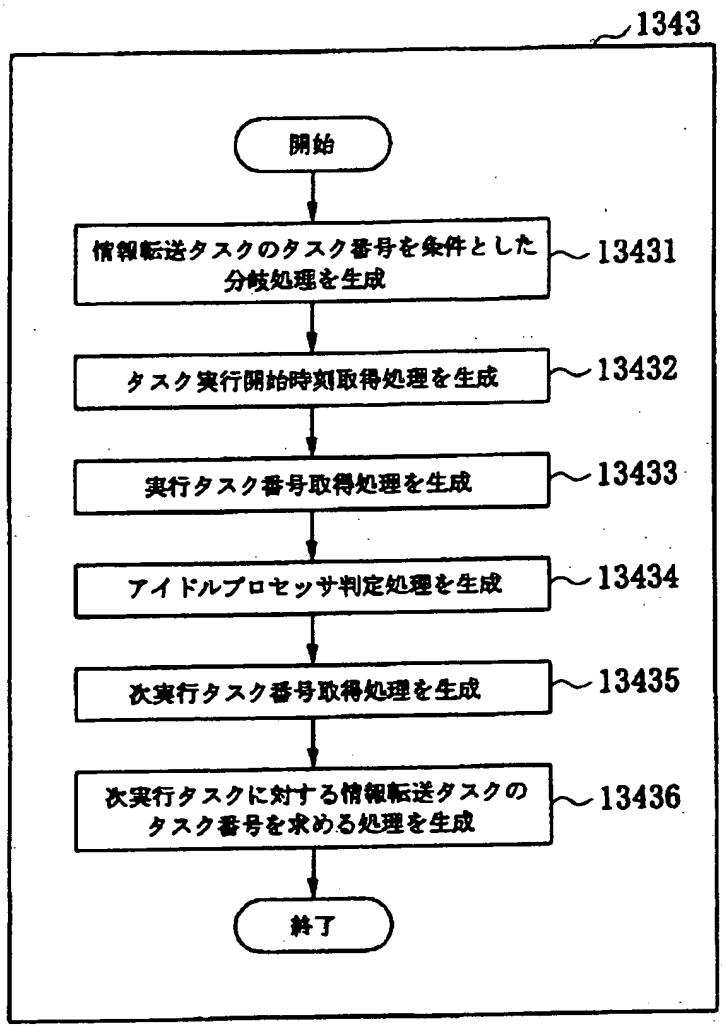
```
46:        case 2:
47:          for(j=1;j<N;j++) {
48:            if(j==1) {b[0] = 0;}
49:            b[j] = a[j] + b[j-1];
50:            if(j==N-1) { printf("b[N-1] = %d¥n",b[N-1]);}
51:          }
52:          break;
53:        case 3:
54:          for(k=1;k<N;k++) {
55:            if(k==1) {c[0] = 0;}
56:            c[k] = a[k] + c[k-1];
57:          }
58:          break;
59:        case 4:
60:          for(m=N-1;m=>0;m--) {
61:            c[m] = c[m+1];
62:            if(m==0) { printf("c[0] = %d¥n",c[0]);}
63:          }
64:          break;
65:        case 5:
66:          break;
67:        case 6:
68:          for(jj=0;jj<N;jj++) {
69:            if( jj!=0) {tmp6 = a[jj];}
70:            tmp6 = b[ jj];
71:          }
72:          break;
73:        case 7:
74:          for(kk=0;kk<N;kk++) {
75:            if( kk!=0) {tmp7 = a[kk];}
76:            tmp7 = c[ kk];
77:          }
78:          break;
79:        case 8:
80:          for(mm=0;mm<N;mm++) {
81:            tmp8 = c[mm];
82:          }
83:          break;
84:      }
85:      if(1<=newMT&&newMT<=NTASK) TData[newMT].Finish = TRUE;
86:      ExecMT[myPE] = INIT_EXEC_MT_NUM;
87:    }
88:#pragma omp end parallel
89:}
```

92

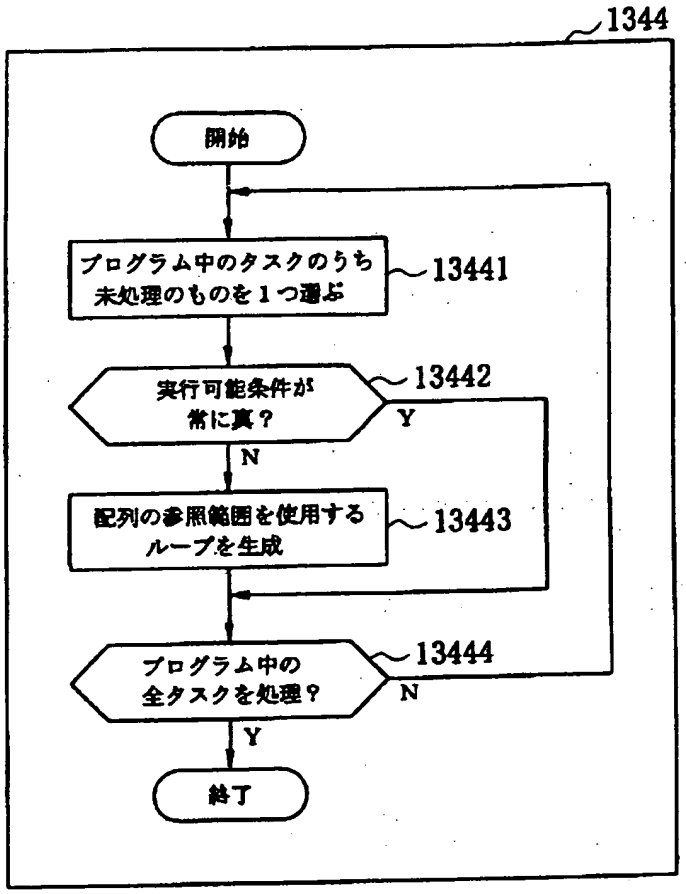
【図10】


133
開始
プログラム中のタスクのうち
未処理のものを1つ選ぶ
1331
実行可能条件が
常に真？
1332
Y
N
タスク内の配列参照範囲を解析して
配列参照範囲テーブルに記録する
1333
タスク実行時間を見積もって
タスクテーブルに記録する
1334
タスクの終了で初めて実行可能条件が
満たされるタスクの総数を求めて
タスクテーブルに記録する
1335
プログラム中の
全タスクを処理？
1336
N
Y
終了

【図12】


1343
開始
情報転送タスクのタスク番号を条件とした分岐処理を生成
13431
タスク実行開始時刻取得処理を生成
13432
実行タスク番号取得処理を生成
13433
アイドルプロセッサ判定処理を生成
13434
次実行タスク番号取得処理を生成
13435
次実行タスクに対する情報転送タスクのタスク番号を求める処理を生成
13436
終了

【図13】


1344
開始
プログラム中のタスクのうち未処理のものを1つ選ぶ
13441
実行可能条件が常に真？
13442
Y
N
配列の参照範囲を使用するループを生成
13443
プログラム中の全タスクを処理？
13444
N
Y
終了